Due: **Friday, May 8, 2009** at the beginning of class.

**Problem 1. Practice with B+ Trees**

For the following questions about B+ trees, you only need to show the final result, but note that if you do this it will be hard to award partial credit if the final result is incorrect. [Note that this problem is about B+ trees that we discussed in lecture, not B-trees. The Weiss textbook also describes B+ trees, though, somewhat confusingly, refers to them as B-trees and then clarifies in a footnote.]

(a) Show the result of inserting 12, 10, 15, 4, 1, 17, 3, 13, and 8 into an initially empty B+ tree with $M = 3$ and $L = 2$.

To maintain consistency in answers, please follow the following rules:

- You should split nodes whenever there is an overflow due to insertion. Another option, discussed in the textbook, is to put a child up for adoption to avoid splitting; this is a viable option, but we ask that you not pursue it and instead split the overflowed node.

- When splitting a leaf node due to insertion overflow, you keep the smallest $\lceil(L+1)/2\rceil$ elements in the original node and put the largest $\lfloor(L+1)/2\rfloor$ elements in the new node. When splitting an internal node, you keep the $\lceil(M+1)/2\rceil$ children with the smaller values attached to the original node and attach the $\lfloor(M+1)/2\rfloor$ children with the larger values to the new node. So, after splitting a node into a “left” node and a “right” node, the number of elements (or children) in the left node will always be greater than or equal to the number of elements (or children) in the right node.

(b) Now show the result of deleting 12, 13, and 15.

**Problem 2. Practice with Hash Functions**

Weiss 5.10.

You should rewrite the `Employee` class in such a way that the `hashCode` method caches the computed value between consecutive calls when possible. You will need accessors (`get` methods) and mutators (`set` methods) for each of the fields, along with the new `hashCode` method.

You may leave out the constructor if you wish, although you should make sure any fields you add are properly initialized. For reference, our solution is about 40 lines of code.
Problem 3. Practice with Hashing

This will give you a chance to get some practice with hash tables, which you will be using in the current project.

(a) Weiss, problem 5.1. Assume the table size is 10.

(b) Weiss, problem 5.2. Choose the new table size to be 19, which is prime and roughly twice as big. Naturally, when rehashing, you should start with the corresponding input hash table and rehash elements from top to bottom; i.e., after hashing, you no longer know the insertion order and should simply iterate through the existing hash table. Also, if any items were not successfully inserted the first time, they should be inserted in this pass, at the end, in the order of failure. (In a real situation, if you failed to insert an element, you would probably rehash to a larger table immediately.)

(c) Suppose a hash table is accessed by open addressing and contains a cell X marked as “deleted”. Suppose that the next successful find hits and moves past cell X and finds the key in cell Y. Suppose we move the found key to cell X, mark cell X as “active” and mark cell Y as “open”. Suppose this policy is used for every find. Would you expect this to work better or worse compared to not modifying the table? Explain your answer.

(d) Suppose that instead of marking cell Y as “open” in the previous question, you mark it as “deleted”. Suppose this policy is used for every find. Would you expect this to work better or worse compared to not modifying the table? Explain your answer.